

14/08/2022

Oque é o NodeJs

Plataforma open-source permite execução da linguagem Javascript do lado do servidor.

V8+libuv_conjuto de módulos.

Arquitetura Event Loop: Call Stack.

Single Thread.

Non-blocking I/O.

API: Interface de Programação de aplicativos.

Conjunto de especificações de possíveis interações entre aplicações.

Documentação para desenvolvedor.

Rest: Transferência representacional de estado. 6 regras.

1 - Client Server.

2- Stateless.

3 – Cache.

4 – Interface Uniform:

Identificação dos recursos.

Representação dos recursos.

Mensagens auto-descritivas.

HATEOAS { "id": 1 };

5 – Camadas.

6 Códigos Sob Demanda.

Métodos de requisições http verbs:

GET – leitura.

POST – Criação.

PUT – Atualização.

DELETE – Deleção.

PATCH – Atualização Parcial.

HTTP Codes:

1xx Informativo – a solicitação foi aceita ou o processo continua em andamento.

2xx Confirmação – 200 Requisição bem sucedida / 201 Created.

3xx Redirecionamento – 301 Moved Permanently / 302 Moved.

4xx Erro do cliente – 400 bad request / 401 Unauthorized / 403 forbidden / 404 not found / unprocessable entity.

5xx erro no servidor – 500 internal server error / 502 bad gateway.

Parâmetros das requisições

Header Params – Parametros que vão no cabeçalho.

Query Params – no final de url, paginação, chave, valor, separação.

Route Params – parâmetros que vão no meio da rota, fazer busca.

Body Params – quando envia no corpo da requisição.

Boas práticas de API REST

A utilização correta dos métodos HTTP.

A utilização correta dos status no retorno das respostas.

Padrão de nomenclatura.

Reduce(); transforma o valores que passarem em um só valor.

Módulo 02

TypeScript: é o Javascript, tipagem estática

```
Class User {
```

```
  Name: string;
```

```
  Username: string;
```

```
  Document: string;
```

```
}
```

```
Const user: User = {
```

```
  Name: "José",
```

```
  Username: "JrSs",
```

```
  Document: "3333"
```

```
}
```

Typescript:

Yarn init -y, yarn add express, yarn add @types/express -D, yarn add typescript, yarn tsc -int,

Converter ts para Js instalar: Yarn add ts-node-dev -D

```
"scripts": {  
  "dev": "ts-node-dev --transpile-only --ignore-watch node_modules --respawn src/server.ts"  
},  
"dependencies": {
```

E dentro do tsconfig.json desabilitar "strict": true.

```
// "strict": true,  
options, */
```

Criando categoria linha 15 (201).send(). falta

```
src > routes > categories.routes.ts > categoriesRoutes.post("/categories") callback  
1 import { Router } from "express";  
2  
3 const categoriesRoutes = Router();  
4  
5 const categories = [];  
6  
7 categoriesRoutes.post("/categories", (request, response) => {  
8   const { name, description } = request.body;  
9  
10  categories.push({  
11    name,  
12    description,  
13  });  
14  
15  return response.status(201);  
16 });  
17  
18 export { categoriesRoutes };  
19
```

```
src > server.ts > ...  
1 import express from "express";  
2 import { categoriesRoutes } from './routes/categories.routes';  
3  
4 const app = express();  
5  
6 app.use(express.json());  
7  
8 app.use(categoriesRoutes)  
9  
10 app.listen(3333, () => console.log("Server is running!"));  
11
```

Inserido Id com uuid yarn add uuid e yarn add @types/uuid

```
import { Router } from "express";  
import { v4 as uuidV4 } from 'uuid' 1.5K (gzipped: 754)
```

```
const category = {  
  name,  
  description,  
  id: uuidV4(),  
};
```

E NO SERVER.TS

```
app.use(express.json());  
app.use("/categories", categoriesRoutes);
```

Inserindo tipagem para categoria

```
src > modules > cars > model > Category.ts > ...  
1  import { v4 as uuidV4 } from "uuid";  
2  
3  // Modelo da classe Category  
4  class Category {  
5  
6      id?: string; // id opcional  
7      name: string;  
8      description: string;  
9      created_at: Date;  
10  
11      // constructor um método que é chamado quando a classe é instanciada, new () iniciar  
12      constructor() {  
13          if (!this.id) { // se ã tiver nem um id desse Category  
14              this.id = uuidV4(); // this id recebe um uuid  
15          }  
16      }  
17  }  
18  
19  export { Category }
```

```
categoriesRoutes.post('/', (request, response) => {  
    const { name, description } = request.body;  
  
    const category = new Category();  
  
    Object.assign(category, {  
        name,  
        description,  
        created_at: new Date()  
    });  
  
    categories.push(category);  
  
    return response.status(201).json({ category });  
});
```

Criando repositório de categoria é como classe responsável pela manipulação de db

```
2  
3  // DTO = Data Transfer Object  
4  export interface ICreateCategoryDTO {  
5      name: string;  
6      description: string;  
7  }
```

```
8  class CategoriesRepository {  
9      private categories: Category[];  
10  
11      constructor() {  
12          this.categories = [];  
13      }  
14  
15      create({ description, name }: ICreateCategoryDTO): void {  
16          const category = new Category();  
17  
18          Object.assign(category, {  
19              name,  
20              description,  
21              created_at: new Date(),  
22          });  
23  
24          this.categories.push(category);  
25      }  
26  }
```

E nas rotas

```
3 | import { CategoriesRepository } from "../repositories/CategoriesRepository";
4 |
5 | const categoriesRoutes = Router();
6 | const categoriesRepository = new CategoriesRepository();
7 |
8 | categoriesRoutes.post("/", (request, response) => {
9 |   const { name, description } = request.body;
10 |
11 |   categoriesRepository.create({ name, description });
12 |
13 |   return response.status(201).send();
14 | });
15 |
16 | export { categoriesRoutes };
```

Listando as categorias no category repository

```
list(): Category[] { // função q retorna a lista de Category
  return this.categories;
}
```

Na rotas

```
categoriesRoutes.get("/", (request, response) => {
  const all = categoriesRepository.list();

  return response.json(all);
});
```

Validado o cadastro de categorias no category repository

```
findByName(name: string): Category { // função de verificação de duplicação do nome
  // find percorre o array procurando pelo nome e retorna
  const category = this.categories.find(category => category.name === name);
  return category;
}
```

Nas rotas

```
categoriesRoutes.post("/", (request, response) => {
  const { name, description } = request.body;

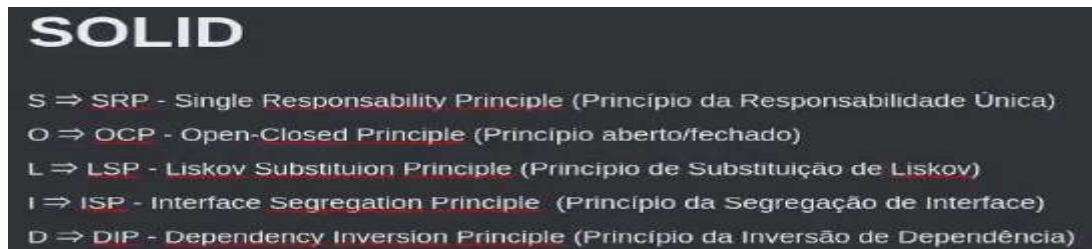
  const categoryAlreadyExists = categoriesRepository.findByName(name);

  if (categoryAlreadyExists) {
    return response.status(400).json({ error: "Category Already exists!" });
  }

  categoriesRepository.create({ name, description });

  return response.status(201).send();
});
```

Repositório: é uma classe, uma camada responsável pela manipulação de dados da aplicação, com banco de dados.



SRP Single Responsibility Principle Princípio de responsabilidade única

Rota só é responsável para receber a requisição e retorna as informações recebida.

Services: É um intermediário entre as rotas e os repositórios, responsável pela regra de negócio.

As rotas existem apenas unicamente para atender as requisições.

Os repositórios existem apenas unicamente para fazer operações para guardar e prover só dados persistidos.

DIP Dependency inversion princípio Princípio de inversão de dependência.

O código que implementa uma política de alto nível não deve depender de código que depende do código que implementa código de baixo nível.

O service não tem que reconhecer qual o tipo do repositório, service o alto nível.

Rotas baixo nível.

LSP Liskov Substitution Principle Princípio de substituição de liskov.

Utilizando o princípio de responsabilidade

```
src > modules > cars > useCases > createCategory > CreateCategoryUseCase.ts > ...
1  import { ICategoriesRepository } from "../../repositories/ICategoriesRepository";
2
3  interface IRequest {
4    name: string;
5    description: string;
6  }
7
8
9
10
11
12
13
14
15  class CreateCategoryUseCase {
16    // precisa do private para acessar
17    constructor(private categoriesRepository: ICategoriesRepository) {}
18
19    execute({ name, description }: IRequest): void {
20      // pega o categoriesRepository e procura pelo name
21      const categoryAlreadyExists = this.categoriesRepository.findByName(name);
22
23      if (categoryAlreadyExists) { // se já existir da um error
24        throw new Error("Category already exists!"); // sempre q tiver error dentro do service
25      }
26
27      this.categoriesRepository.create({ name, description }); // aqui chama o
      categoriesRepository
28    }
29  }
30
31  export { CreateCategoryUseCase };
```


Nas rotas

```
const categoriesRoutes = Router();
const categoriesRepository = new CategoriesRepository();

categoriesRoutes.post("/", (request, response) => {
  const { name, description } = request.body;

  const createCategoryService = new CreateCategoryService(categoriesRepository);

  createCategoryService.execute({ name, description });

  return response.status(201).send();
});
```

Utilizando o princípio da substituição de Liskov

```
src > modules > cars > repositories > ICategoriesRepository.ts > ICategoriesRepository > list
1  import { Category } from "../model/Category";
2
3  // DTO = Data Transfer Object
4  export interface ICreateCategoryDTO {
5    name: string;
6    description: string;
7  }
8
9  interface ICategoriesRepository {
10   findByName(name: string): Category;
11   list(): Category[];
12   create({ name, description }: ICreateCategoryDTO):void ;
13 }
14
15 export { ICategoriesRepository };
```

Em categories repository

```
// Responsável por cuidar das funcionalidade ao banco de dados etc
class CategoriesRepository implements ICategoriesRepository {
  private categories: Category[]; // do tipo array de Category, só o categories tem acesso
```

Criando servisse de especificação e separando em módulos

```
src > modules > cars > model > Specifications.ts > ...
1  import { v4 as uuidv4 } from "uuid";
2
3  class Specifications {
4    id?: string; // id opcional
5    name: string;
6    description: string;
7    created_at: Date;
8
9    // constructor um método que é chamado quando a classe é instanciada, new () iniciar
10   constructor() {
11     if (!this.id) { // se ã tiver nem um id desse Category
12       this.id = uuidv4(); // this id recebe um uuid
13     }
14   }
15 }
16
17 export { Specifications };
```

Módulos: pequenos blocos da aplicação.

Criando repostório de especificação

```
src > modules > cars > repositories > ISpecificationsRepository.ts > ...
1  import { Specifications } from "../../model/Specifications";
2
3  interface ICreateSpecificationsDTO {
4    name: string;
5    description: string;
6  }
7
8  interface ISpecificationsRepository {
9    create({ name, description }: ICreateSpecificationsDTO): void;
10   findByName(name: string): Specifications;
11 }
12
13 export { ISpecificationsRepository, ICreateSpecificationsDTO };
```

Specification repository

```
src > modules > cars > repositories > implementations > SpecificationsRepository.ts > ...
1  import { Specifications } from "../../model/Specifications";
2  import { ICreateSpecificationsDTO, ISpecificationsRepository } from "../ISpecificationsRepository";
3
4  class SpecificationRepository implements ISpecificationsRepository {
5    private specifications: Specifications[];
6
7    constructor() {
8      this.specifications = [];
9    }
10
11    create({ name, description }: ICreateSpecificationsDTO): void {
12      const specification = new Specifications();
13
14      Object.assign(specification, {
15        name,
16        description,
17        created_at: new Date(),
18      });
19
20      this.specifications.push(specification);
21    }
22
23    findByName(name: string): Specifications {
24      const specification = this.specifications.find(specification => specification.name === name);
25
26      return specification;
27    }
28  }
29
30 export { SpecificationRepository };
```

CreateSpecification Service

```
src > modules > cars > services > CreateSpecificationService.ts > ...
1  import { ISpecificationsRepository } from "../repositories/ISpecificationsRepository";
2
3  interface IRequest {
4    name: string;
5    description: string;
6  }
7
8  class CreateSpecificationService {
9    constructor(private specificationsRepository: ISpecificationsRepository) {}
10   execute({ name, description }: IRequest): void {
11     const specificationAlreadyExists = this.specificationsRepository.findByName(
12       name
13     );
14
15     if (specificationAlreadyExists) {
16       throw new Error("Specification already exists!");
17     }
18
19     this.specificationsRepository.create({
20       name,
21       description,
22     });
23   }
24 }
25
26 export { CreateSpecificationService };
```


Rotas specifications

```
src > routes > specifications.routes.ts > specificationsRoutes.post("/").callback
3 import { CreateSpecificationService } from "modules/cars/services/CreateSpecificationService";
4
5 const specificationsRoutes = Router();
6
7 const specificationsRepository = new SpecificationsRepository();
8
9 specificationsRoutes.post("/", (request, response) => {
10   const { name, description } = request.body;
11   const createSpecificationService = new CreateSpecificationService(
12     specificationsRepository
13   );
14
15   createSpecificationService.execute({ name, description });
16
17   return response.status(201).send();
18 });
19
20 export { specificationsRoutes };
```

Index da rotas

```
src > routes > index.ts > ...
1 import { Router } from "express";
2
3 import { categoriesRoutes } from "../categories.routes";
4 import { specificationsRoutes } from "../specifications.routes";
5
6 const router = Router();
7
8 router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
9 router.use("/specifications", specificationsRoutes);
```

Criando Usecade de categoria

UseCases: as regras de negócios da aplicação.

Controllers: são classes que recebe a requisição, e retorna a resposta para quem está chamando. Responsável pelas requisições e respostas.

```
5 class CreateCategoryUseCase {
6   // precisa do private para acessar
7   constructor(private categoriesRepository: ICategoriesRepository) {}
8
9   execute({ name, description }: IRequest): void {
10     // pega o categoriesRepository e procura pelo nome
11   }
12 }
```

```
src > modules > cars > useCases > createCategory > CreateCategoryController.ts > ...
1 import { Request, Response } from "express";
2
3 import { CreateCategoryUseCase } from "../CreateCategoryUseCase";
4
5 class CreateCategoryController {
6   constructor(private createCategoryUseCase: CreateCategoryUseCase) {}
7
8   handle(request: Request, response: Response): Response {
9     const { name, description } = request.body; // pega o nome, descrição do body
10
11     this.createCategoryUseCase.execute({ name, description });
12
13     // status 201 algo criado com sucesso, se não manda json manda send
14     return response.status(201).send();
15   }
16 }
17
18 export { CreateCategoryController };
```

```
src > modules > cars > useCases > createCategory > index.ts > categoriesRepository
1 import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
2 import { CreateCategoryController } from "../CreateCategoryController";
3 import { CreateCategoryUseCase } from "../CreateCategoryUseCase";
4
5 const categoriesRepository = CategoriesRepository.getInstance();
6 const createCategoryUseCase = new CreateCategoryUseCase(categoriesRepository);
7 const createCategoryController = new CreateCategoryController(createCategoryUseCase);
8
9 export { createCategoryController };
```

```

15 // rota de criar categorias
16 categoriesRoutes.post("/", (request, response) => {
17   return createCategoryController.handle(request, response);
18 });
19
20 // rota de listar categoria
21 categoriesRoutes.get("/", (request, response) => {
22   return listCategoriesController.handle(request, response)
23 });

```

Refatorando a listagem de categoria

```

src > modules > cars > useCases > listCategories > ListCategoriesUseCase.ts > execute
1 import { Category } from '../../model/Category';
2 import { ICategoriesRepository } from '../../repositories/ICategoriesRepository';
3
4 class ListCategoriesUseCase {
5   // precisa do private para acessar
6   constructor(private categoriesRepository: ICategoriesRepository) {}
7
8   execute(): Category[] { // retorna lista de categoria
9     const categories = this.categoriesRepository.list();
10
11     return categories;
12   }
13 }
14
15 export { ListCategoriesUseCase };

```

```

src > modules > cars > useCases > listCategories > ListCategoriesController.ts > ...
1 import { Request, Response } from "express";
2
3 import { ListCategoriesUseCase } from "../ListCategoriesUseCase";
4
5 class ListCategoriesController {
6   constructor(private listCategoriesUseCase: ListCategoriesUseCase) {}
7
8   handle(request: Request, response: Response): Response {
9     const all = this.listCategoriesUseCase.execute(); // chama a lista
10
11     return response.json(all);
12   }
13 }
14
15 export { ListCategoriesController };

```

```

src > modules > cars > useCases > listCategories > index.ts > ...
1 import { ListCategoriesUseCase } from "../ListCategoriesUseCase";
2 import { ListCategoriesController } from "../ListCategoriesController";
3 import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
4
5 const categoriesRepository = CategoriesRepository.getInstance();
6 const listCategoriesUseCase = new ListCategoriesUseCase(categoriesRepository);
7 const listCategoriesController = new ListCategoriesController(listCategoriesUseCase);
8
9 export { listCategoriesController };

```

Conhecendo Singleton Pattern instância de uma classe que vai ser uma instância global. No categoriesRepository

```

class CategoriesRepository implements ICategoriesRepository {

    private categories: Category[]; // do tipo array de Category, só o categories tem acesso

    private static INSTANCE: CategoriesRepository; // para listar os produtos

    private constructor() {
        this.categories = []; // aqui que cria o categories
    }

    public static getInstance(): CategoriesRepository {
        if (!CategoriesRepository.INSTANCE) { //se não tem valores atribuido
            CategoriesRepository.INSTANCE = new CategoriesRepository();
        }

        return CategoriesRepository.INSTANCE; //se já tem valores manda ele
    }
}

```

No index do useCase do createcategory e no list

```

2 import { CreateCategoryController } from "../createcategorycontroller";
3 import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
4
5 const categoriesRepository = CategoriesRepository.getInstance();

```

Criando use case de especificação

```

src > modules > cars > useCases > createSpecifications > CreateSpecificationsController.ts > ...
1 import { Request, Response } from "express";
2
3 import { CreateSpecificationsUseCase } from "../CreateSpecificationsUseCase";
4
5 class CreateSpecificationsController {
6     constructor(private createSpecificationsUseCase: CreateSpecificationsUseCase) {}
7
8     handle(request: Request, response: Response): Response {
9         const { name, description } = request.body;
10
11         this.createSpecificationsUseCase.execute({ name, description });
12
13         return response.status(201).send();
14     }
15 }
16
17 export { CreateSpecificationsController };

```

```

src > modules > cars > useCases > createSpecifications > CreateSpecificationsUseCases.ts > ...
1 import { ISpecificationsRepository } from "../../../repositories/ISpecificationsRepository";
2
3 interface IRequest {
4     name: string;
5     description: string;
6 }
7
8 class CreateSpecificationsUseCase {
9     constructor(private specificationsRepository: ISpecificationsRepository) {}
10
11     execute({ name, description }: IRequest): void {
12         const specificationsAlreadyExists = this.specificationsRepository.findByName(name);
13
14         if (specificationsAlreadyExists) {
15             throw new Error("Specification already exists!");
16         }
17
18         this.specificationsRepository.create({
19             name,
20             description,
21         });
22     }
23 }
24
25 export { CreateSpecificationsUseCase };

```

```
src > modules > cars > useCases > createSpecifications > index.ts > ...
1 import { CreateSpecificationsUseCase } from "../CreateSpecificationsUseCase";
2 import { CreateSpecificationsController } from "../CreateSpecificationsController";
3 import { SpecificationRepository } from "../../repositories/implementations/SpecificationsRepository";
4
5 const specificationsRepository = new SpecificationRepository();
6 const createSpecificationsUseCase = new CreateSpecificationsUseCase(specificationsRepository);
7 const createSpecificationsController = new CreateSpecificationsController(createSpecificationsUseCase);
8
9 export { createSpecificationsController };
```

Refatorando as rotas criar um index dentro das rotas

```
src > routes > index.ts > ...
1 import { Router } from "express";
2
3 import { categoriesRoutes } from "../categories.routes";
4 import { specificationsRoutes } from "../specifications.routes";
5
6 const router = Router();
7
8 router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
9 router.use("/specifications", specificationsRoutes);
10
11 export { router };
```

Trabalhando com Upload

Upload de arquivos. Usa o multer é como um middleware dentro da rota.

[Yarn add multer](#)

[Yarn add @types/multer -D](#)

Criar pasta tmp. E nas rotas de categories coloca a configuração do multer.

```
const upload = multer({
  dest: "./tmp",
});
```

Single por quê só quer upload de um arquivo, e dentro do single coloca um nome.

Ler o arquivo.

```
categoriesRoutes.post("/import", upload.single("file"), (request, response) => {
  const { file } = request;
  console.log(file);
  return response.send();
});
```

Criando o use case para importar categorias

```
src > modules > cars > useCases > importCategory > index.ts > ...
1 import { CategoriesRepository } from "../../repositories/implementations/CategoriesRepository";
2 import { ImportCategoryController } from "../ImportCategoryController";
3 import { ImportCategoryUseCase } from "../ImportCategoryUseCase";
4
5 const categoriesRepository = CategoriesRepository.getInstance();
6 const importCategoryUseCase = new ImportCategoryUseCase(categoriesRepository);
7 const importCategoryController = new ImportCategoryController(importCategoryUseCase);
8
9 export { importCategoryController };
```

```

src > modules > cars > useCases > importCategory > ImportCategoryController.ts > ...
1  import { Request, Response } from "express";
2
3  import { ImportCategoryUseCase } from "../ImportCategoryUseCase";
4
5  class ImportCategoryController {
6      constructor(private importCategoryUseCase: ImportCategoryUseCase) {}
7
8      handle(request: Request, response: Response): Response {
9          const { file } = request;
10
11          this.importCategoryUseCase.execute(file);
12
13          return response.send();
14      }
15  }
16
17  export { ImportCategoryController };

```

Conhecendo conceito de streaming

READFILE faz a leitura tudo de uma vez do arquivo, **streaming**: permite ler arquivos por partes, sem precisar consumir muita memória.

FS, a função `createReadStream()`, permite que faça a leitura do arquivo em ponto.

E recebe o path do arquivo. Usa o **PIPE**, pega o streaming que está sendo lido e dentro dele, ele joga o que foi lido para o lugar que a gente determinar.

LIB CSV PARSE: [Yarn add csv-parse](#)

```

src > modules > cars > useCases > importCategory > ImportCategoryUseCases.ts > ImportCategoryUseCase > loadCategories > <function>
1  import fs from "fs";
2  import { parse } from "csv-parse";
3
4  import { ICategoriesRepository } from "../../repositories/ICategoriesRepository";
5
6  interface IImportCategory {
7      name: string;
8      description: string;
9  }
10
11  class ImportCategoryUseCase {
12      constructor(private categoriesRepository: ICategoriesRepository) {}
13
14
15      loadCategories(file: Express.Multer.File): Promise<IImportCategory[]> {
16          return new Promise((resolve, reject) => {
17              const stream = fs.createReadStream(file.path); // faz leitura em parte
18              const categories: IImportCategory[] = [];
19
20              const parseFile = parse();
21
22              // pipe pega o que foi lido e joga pra onde nos quer
23              stream.pipe(parseFile);
24
25              parseFile.on("data", async (line) => {
26                  const [name, description] = line;
27                  categories.push({ name, description });
28              })
29              .on("end", () => {
30                  fs.promises.unlink(file.path); // para remover os arquivos do tmp
31                  resolve(categories);
32              })
33              .on("error", (err) => {
34                  reject(err);
35              });
36          });
37      }

```



```

38
39 ✓ async execute(file: Express.Multer.File): Promise<void> {
40   const categories = await this.loadCategories(file);
41
42   categories.map(async (category) => {
43     const { name, description } = category;
44
45     const existCategory = this.categoriesRepository.findByName(name); // existe um categoria
46
47     if (!existCategory) { // se ã tem cria
48       this.categoriesRepository.create({
49         name,
50         description,
51       });
52     }
53   });
54 }
55 }
56
57 export { ImportCategoryUseCase };

```

SWAGGER DOCUMENTAÇÃO: [yarn add swagger-ui-express](#)

[Yarn add @types/swagger-ui-express -D](#)

Criar dentro do server, 1 a rota da documentação, 2 chama o server do Swagger, 3 o setup é o arquivo json onde está as informações da aplicação.

Para resolver o problema de importação do Swagger.

```

| resolving a module. */
| "resolveJsonModule": true,
| // "noResolve": true,

```

```

import swaggerUi from "swagger-ui-express";

import swaggerFile from "../swagger.json";

import { router } from "../routes";

const app = express();

app.use(express.json());

app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerFile))

app.use(router);

```

DOCKER

- Ferramenta para criação de containers
- Container: Ambiente isolado
- Imagens: Instruções para criação de um container
- O que "roda" localmente "roda" em produção
- Mesmo SO, compartilhando recursos da máquina host

Workdir: define uma pasta onde as informações estejam contidas, /usr/app

COPY para copiar as dependências

RUN instala as dependências no Docker

Comando de rodar o Docker: **Docker build -t rentx .**

```
Dockerfile > ...
1 FROM node
2
3 WORKDIR /usr/app
4
5 COPY package.json ./
6
7 RUN npm install
8
9 COPY . .
10
11 EXPOSE 3333
12
13 CMD ["npm", "run", "dev"]
```

```
.dockerignore
1 node_modules
2 .git
3 .vscode
```

```
version: "3.7"

services:
  app:
    build: .
    container_name: rentx
    ports:
      - 3333:3333
    volumes:
      - ./usr/app
```

Docker ps = ver os container rodando

Docker run -p 3333:3333 rentx = rodar a imagem

Docker compose é um orquestrador de container

Docker-compose up = para builda e executa em real time, e de da -d roda em background

Docker logs nome -f = mostra toda execução em real time.

Comandos docker

Docker ps = mostra lista dos container de pé

Docker ps -a = lista todo container de pé e que não estão

Docker rm name-id = remover container

Docker start name = inicia o container

Docker stop id-container = para o container

Docker-compose up -D = sobe o container em background

Docker-compose stop = para o Docker compose

Docker-compose down = remove tudo que tiver criado dentro do serviço

Docker exec -it namecontainer /bin/bash = acessa o container.

Docker logs nameimage -f = acompanhar os logs

Docker-compose up --force-recreate -d = força recria novamente

Banco de dados Postgres

[Yarn add typeorm reflect-metadata](#)

[Yarn add pg](#)

Desabilitar no ts-config

```
// "jsx": "preserve",  
"experimentalDecorators": true,  
draft decorators. */  
"emitDecoratorMetadata": true,  
declarations in source files. */
```

E criam src database uma pasta, e um index.ts

Criar no Docker a image postgres

```
import { DataSource } from "typeorm";  
const dataSource = new DataSource({  
  type: "postgres",  
  host: "localhost",  
  port: 5432,  
  username: "docker",  
  password: "ignite",  
  database: "rentx",  
});  
  
dataSource.initialize();  
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'  
database_ignite
```

```
ormconfig.json > ...
1 {
2   "type": "postgres",
3   "port": 5432,
4   "host": "localhost",
5   "username": "docker",
6   "password": "ignite",
7   "database": "rentx",
8   "migrations": ["./src/database/migrations/*.ts"],
9   "entities": ["./src/modules/**/*.entities/*.ts"],
10  "cli": {
11    "migrationsDir": "./src/database/migrations"
12  }
13 }
```

```
docker-compose.yml
1  version: "3.9"
2
3  services:
4    database:
5      image: postgres
6      container_name: database_ignite
7      restart: always
8      ports:
9        - 5432:5432
10     environment:
11       - POSTGRES_USER=docker
12       - POSTGRES_PASSWORD=ignite
13       - POSTGRES_DB=rentx
14     volumes:
15       - pgdata:/data/postgres
16
17
18     app:
19       build: .
20       container_name: rentx
21       restart: always
22       ports:
23         - 3333:3333
24         - 9229:9229
25       volumes:
26         - ./usr/app
27       links:
28         - database
29       depends_on:
30         - database
31
32
33  volumes:
34    pgdata:
35      driver: local
```

typeorm

<https://www.notion.so/Refatora-o-Docker-com-TypeORM-4500fc0d075349ac9b97d670e734d41b>

<https://efficient-sloth-d85.notion.site/Atualizando-o-TypeORM-no-Rentx-7988bcb23f9f417197fcf2113a74161a>

github: <https://github.com/rocketseat-education/ignite-nodejs-rentx>

```
"typeorm": "^0.2.31",
"pg": "^8.5.1",
```

```
"reflect-metadata": "^0.1.13",
```

Migrations

Yarn typeorm migration:create -n CreateCategories

Criar pasta migrations dentro de database e no ormconfig e no package em script

```
ts ,  
"typeorm": "ts-node-dev ./node_modules/typeorm/cli"
```

```
"cli": {  
  "migrationsDir": "./src/database/migrations"  
}
```

```
import {MigrationInterface, QueryRunner, Table} from "typeorm";  
  
export class CreateCategories1669381715993 implements MigrationInterface {  
  public async up(queryRunner: QueryRunner): Promise<void> {  
    await queryRunner.createTable(  
      new Table({  
        name: "categories",  
        columns: [  
          {  
            name: "id",  
            type: "uuid",  
            isPrimary: true,  
          },  
          {  
            name: "name",  
            type: "varchar",  
          },  
          {  
            name: "description",  
            type: "varchar",  
          },  
          {  
            name: "created_at",  
            type: "timestamp",  
            default: "now()",  
          },  
        ],  
      })  
    );  
  }  
  
  public async down(queryRunner: QueryRunner): Promise<void> {  
    await queryRunner.dropTable("categories");  
  }  
}
```

Down se da errado ele deleta a tabela

Depois no terminal

Yarn typeorm migration:run

Criando entidade de categoria

```

modules > cars > entities > Category.ts > ...
1  import { v4 as uuidv4 } from "uuid";
2  import { Entity, Column, PrimaryColumn, CreateDateColumn } from "typeorm";
3
4  @Entity("categories")
5  // Modelo da classe Category
6  class Category {
7
8      @PrimaryColumn()
9      id?: string; // id opcional
10
11      @Column()
12      name: string;
13
14      @Column()
15      description: string;
16
17      @CreateDateColumn()
18      created_at: Date;
19
20      // constructor um método que é chamado quando a classe é instanciada, new () iniciar
21      constructor() {
22          if (!this.id) { // se ã tiver nem um id desse Category
23              this.id = uuidv4(); // this id recebe um uuid
24          }
25      }
26  }
27
28  export { Category }

```

No seu **package.json**, adicione o script do typeorm:

```
"typeorm": "ts-node-dev -r tsconfig-paths/register ./node_modules/typeorm/cli.js -d src/database/index.ts"
```

No arquivo **index.ts** da pasta **database**, altere o conteúdo para:

```
import { DataSource } from "typeorm";
```

```
const AppDataSource = new DataSource({
```

```
    type: "postgres",
```

```
    host: "localhost",
```

```
    port: 5432,
```

```
    username: "docker",
```

```
    password: "ignite",
```

```
    database: "rentx",
```

```
});
```

```
export function createConnection(host = "database_ignite"): Promise<DataSource> {  
  
  return AppDataSource.setOptions({ host }).initialize();  
  
}
```

```
export default AppDataSource;
```

No arquivo **server.ts**, mude a importação do banco para:

```
import { createConnection } from "../database";
```

```
createConnection();
```

Por fim, atualize o seu **docker-compose.yml** para:

```
version: "3.9"
```

```
services:
```

```
  database_ignite:
```

```
    image: postgres
```

```
    container_name: database_ignite
```

```
    restart: always
```

```
    ports:
```

```
      - 5432:5432
```

```
    environment:
```

```
      - POSTGRES_USER=docker
```


- POSTGRES_PASSWORD=ignite

- POSTGRES_DB=rentx

volumes:

- pgdata:/data/postgres

app:

build: .

container_name: rentx

restart: always

ports:

- 3333:3333

volumes:

- ./usr/app

links:

- database_ignite

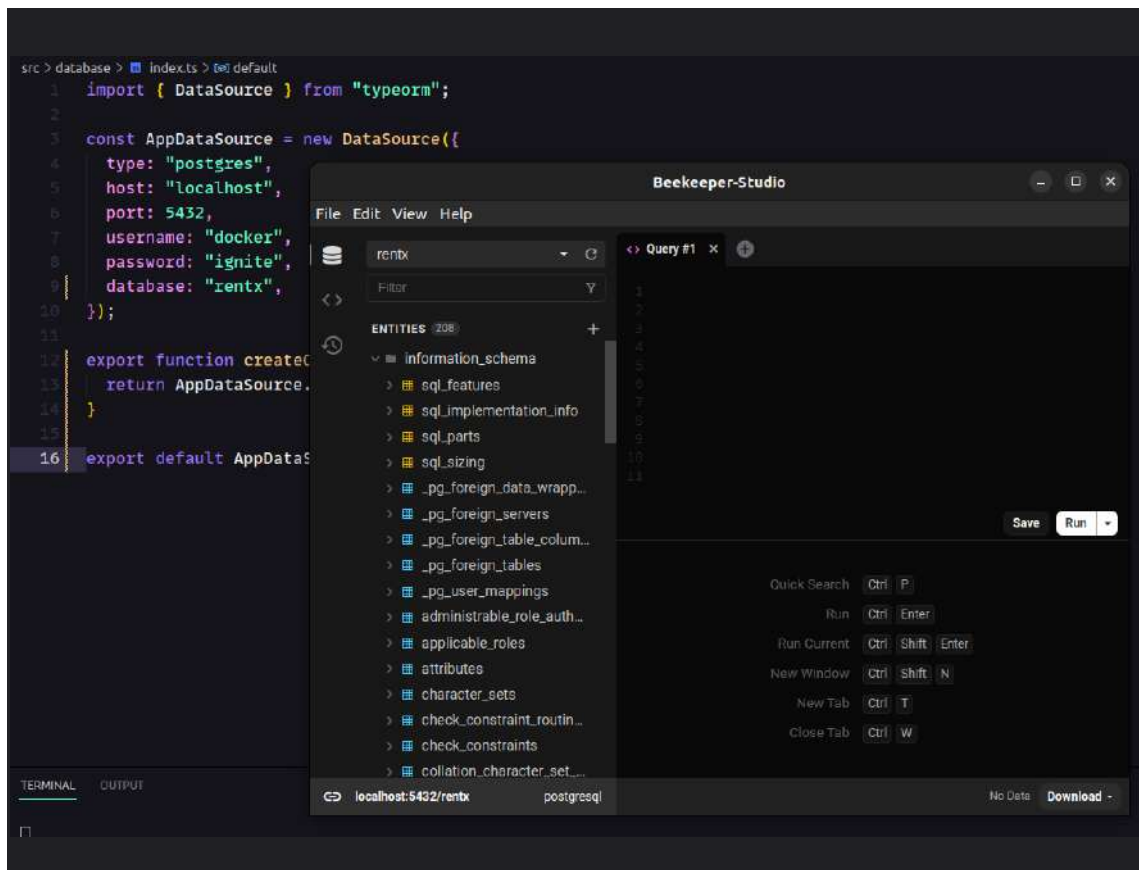
depends_on:

- database_ignite

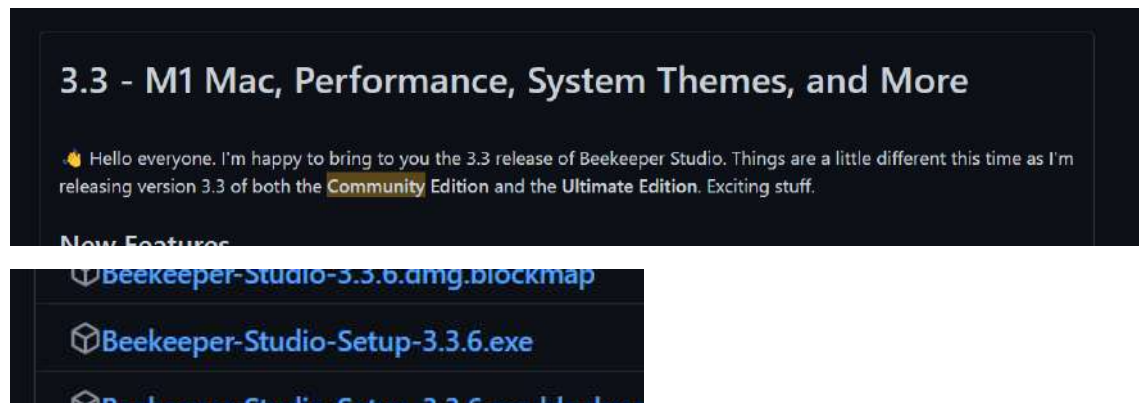
volumes:

pgdata:

driver: local



<https://github.com/beekeeper-studio/beekeeper-studio/releases?q=Community+&expanded=true>



```

3
4 @Entity("categories")
5 // Modelo da classe Category
6 class Category {
7
8     @PrimaryKey()
9     id?: string; // id opcional
10
11     @Column()
12     name: string;
13
14     @Column()
15     description: string;
16
17     @CreateDateColumn()
18     created_at: Date;
19
20     // constructor um método que é chamado quando a entidade é criada
21     constructor() {
22         if (!this.id) { // se não tiver nenhum id
23             this.id = uuidv4(); // this id é gerado automaticamente
24         }
25     }
26 }

```

```

private repository: Repository<Category>; // acesso só internamente

private static INSTANCE: CategoriesRepository; // para listar os produtos

private constructor() {
    this.repository = getRepository(Category);
}

public static getInstance(): CategoriesRepository {
    if (!CategoriesRepository.INSTANCE) {
        CategoriesRepository.INSTANCE = new CategoriesRepository();
    }

    return CategoriesRepository.INSTANCE;
}

```

```

async create({ name, description}: ICreateCategoryDTO): Promise<void> { // função sem retorno
    const category = this.repository.create({
        description,
        name,
    }); // cria a entidade para poder salvar

    // manda pro categories name, descrição, uuid
    await this.repository.save(category)
}

async list(): Promise<Category[]> { // função q retorna a lista de Category
    const categories = await this.repository.find(); // retorna uma lista
    return categories;
}

async findByName(name: string): Promise<Category> { // função de verificação de duplicação do nome
    // SELECT * FROM categories WHERE name = "name" LIMIT 1
    const category = await this.repository.findOne({ name })
    return category;
}
}

```

```
interface ICategoriesRepository {
  findByName(name: string): Promise<Category>;
  list(): Promise<Category[]>;
  create({ name, description }: ICreateCategoryDTO): Promise<void> ;
}

export { ICategoriesRepository };
```

Yarn typeorm migration:run / cria as tabelas

Yarn typeorm migration:revert / desfaz as tabelas

Injeção de dependência baixar o [yarn add tsyringe](#)

Cria pasta em src shared e container outra pastas dentro de shred.

```
src > shared > container > index.ts
1 import { container } from "tsyringe";
2
3 import { ICategoriesRepository } from "../../modules/cars/repositories/ICategoriesRepository";
4 import { CategoriesRepository } from "../../modules/cars/repositories/implementations/CategoriesRepository";
5
6 container.registerSingleton<ICategoriesRepository>(
7   // da um nome e referencia a classe
8   "CategoriesRepository",
9   CategoriesRepository
10 );
```

No useCase

```
src > modules > cars > useCases > createCategory > CreateCategoryUseCase.ts > ...
1 import { inject, injectable } from "tsyringe";
2
3 import { ICategoriesRepository } from "../../repositories/ICategoriesRepository";
4
5 interface IRequest {
6   name: string;
7   description: string;
8 }
9
10 @injectable()
11 class CreateCategoryUseCase {
12   // precisa do private para acessar
13   constructor(
14     @inject("CategoriesRepository")
15     private categoriesRepository: ICategoriesRepository) {}
16
17   async execute({ name, description }: IRequest): Promise<void> {
18     // pega o categoriesRepository e procura pelo name
19     const categoryAlreadyExists = await this.categoriesRepository.findByName(name);
```

No controller, e remove o index.ts

```

> modules > cars > useCases > createCategory > CreateCategoryController.ts > ...
1  import { Request, Response } from "express";
2  import { container } from "tsyringe";
3
4  import { CreateCategoryUseCase } from "../CreateCategoryUseCase";
5
6  class CreateCategoryController {
7
8
9      async handle(request: Request, response: Response): Promise<Response> {
10         const { name, description } = request.body; // pega o nome, descrição do body
11
12         const createCategoryUseCase = container.resolve(CreateCategoryUseCase);
13
14         await createCategoryUseCase.execute({ name, description });
15
16         // status 201 algo criado com sucesso, se não manda isso, manda cond

```

E nas rotas passa assim

```

1  // ...
2
3  const createCategoryController = new CreateCategoryController();
4
5  // rota de criar categorias
6  categoriesRoutes.post("/", createCategoryController.handle);
7
8  // rota de listar categoria
9

```

E chama no serve o container

```

16 server.ts > ...
1  import "reflect-metadata";
2  import express from "express";
3  import swaggerUi from "swagger-ui-express";
4
5
6
7  import "./database";
8  import "./shared/container";
9

```

Criando Users

Yarn typeorm migration: create -n aidaonome

```

export class CreateUsers1669976709139 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.createTable(
      new Table({
        name: "users",
        columns: [
          {
            name: "id",
            type: "uuid"
          },
          {
            name: "name",
            type: "varchar"
          },
          {
            name: "username",
            type: "varchar",
            isUnique: true
          },
          {
            name: "isAdmin",
            type: "boolean",
            default: false
          },
          {
            name: "created_at",
            type: "timestamp",
            default: "now()"
          }
        ],
      })
    );
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.dropTable("users");
  }
}

```

Depois yarn typeorm migration:run para criar a tabela

Criar pasta account com a entidades user.ts


```

modules > accounts > entities > User.ts > User > constructor
import { v4 as uuidV4 } from "uuid";
import { Column, CreateDateColumn, Entity, PrimaryColumn } from "typeorm";

@Entity("users")
class User {

  @PrimaryColumn()
  id: string;

  @Column()
  name: string;

  @Column()
  username: string;

  @Column()
  email: string;

  @Column()
  password: string;

  @Column()
  driver_license: string;

  @Column()
  isAdmin: boolean;

  @CreateDateColumn()
  created_at: Date;

  constructor() {
    if (!this.id) {
      this.id = uuidV4();
    }
  }
}

```

Criando repositórios. Dentro IUserrepository e pasta implementations e dentro UserRepository

```

modules > accounts > repositories > IUserRepository.ts > IUserRepository > create
1 import { ICreateUserDTO } from "../dtos/ICreateUserDTO";
2
3
4 interface IUserRepository {
5   create(data: ICreateUserDTO): Promise<void>;
6 }
7
8 export { IUserRepository }

```

```

modules > accounts > repositories > implementations > UsersRepository.ts > ...
1 import { Repository, getRepository } from "typeorm";
2
3 import { ICreateUserDTO } from "../../dtos/ICreateUserDTO";
4 import { IUsersRepository } from "../IUsersRepository";
5 import { User } from "../../entities/User";
6
7
8 class UsersRepository implements IUsersRepository {
9     private repository: Repository<User>
10
11     constructor() {
12         this.repository = getRepository(User);
13     }
14
15     async create({ name, password, username, email, driver_license }: ICreateUserDTO): Promise<void> {
16         const user = this.repository.create({
17             name, password, username, email, driver_license
18         });
19         await this.repository.save(user);
20     }
21 }
22
23 export { UsersRepository }

```

Cria pasta DTO

```

src > modules > accounts > dtos > ICreateUserDTO.ts > ICreateUserDTO
1 export interface ICreateUserDTO {
2     name: string;
3     username: string;
4     password: string;
5     email: string;
6     driver_license: string;
7 }

```

Cria pasta useCases e dentro outra pasta createUser e dentro a createUserUseCase.ts e createUserController.ts

```

modules > accounts > useCases > createUser > createUserUseCase.ts > ...
1 import { inject, injectable } from "tsyringe";
2
3 import { ICreateUserDTO } from "../../dtos/ICreateUserDTO";
4 import { IUsersRepository } from "../../repositories/IUsersRepository";
5
6
7 @injectable()
8 class CreateUserUseCase {
9
10     constructor(
11         @inject("UsersRepository")
12         private usersRepository: IUsersRepository
13     ) {}
14
15     async execute({ name, username, password, email, driver_license }: ICreateUserDTO): Promise<void> {
16         await this.usersRepository.create({
17             name, username, password, email, driver_license
18         });
19     }
20 }
21
22 export { CreateUserUseCase }

```

Controller

```

> modules > accounts > useCases > createUser > CreateUserController.ts > CreateUserController > handle
1  import { Request, Response } from "express";
2  import { container } from "tsyringe";
3
4  import { CreateUserUseCase } from "../CreateUserUseCase";
5
6
7  class CreateUserController {
8      async handle(request: Request, response: Response): Promise<Response> {
9          const { name, username, email, password, driver_license } = request.body;
10         const createUserUseCase = container.resolve(CreateUserUseCase);
11
12         await createUserUseCase.execute({
13             name, username, email, password, driver_license
14         });
15
16         return response.status(201).send();
17     }
18 }
19
20 export { CreateUserController }

```

E no container cria

```

21 container.registerSingleton<IUsersRepository>({
22     "UsersRepository",
23     UsersRepository
24 });

```

Rotas agora

```

src > routes > users.routes.ts > ...
1  import { Router } from "express";
2  import { CreateUserController } from "../../modules/accounts/useCases/createUser/CreateUserController";
3
4  const usersRoutes = Router();
5
6  const createUserController = new CreateUserController();
7
8  usersRoutes.post("/", createUserController.handle);
9
10 export { usersRoutes };

```

E no index da rota

```

src > routes > index.ts > ...
1  import { Router } from "express";
2
3  import { categoriesRoutes } from "../categories.routes";
4  import { specificationsRoutes } from "../specifications.routes";
5  import { usersRoutes } from "../users.routes";
6
7  const router = Router();
8
9  router.use("/categories", categoriesRoutes); // cria a rota categories path inicia com categories
10 router.use("/specifications", specificationsRoutes);
11 router.use("/users", usersRoutes);
12
13 export { router };

```

Alterar tabelas yarn typeorm migration:create -n AlterUserDeleteUsername

```
import { MigrationInterface, QueryRunner, TableColumn } from "typeorm";

export class AlterUserDeleteUsername1669979467860 implements MigrationInterface {

    public async up(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.dropColumn("users", "username");
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.addColumn("users",
            new TableColumn({
                name: "username",
                type: "varchar"
            })
        );
    }
}
```

Criptografia de senha. [Yarn add bcryptjs yarn add @types/bcryptjs -D](#)

Recebe a senha e o salt que o salt melhor e o 8

```
16
17     async execute({ name, password, email, driver_license }: ICreateUserDTO): Promise<void> {
18         const passwordHash = await hash(password, 8);
19
20         await this.usersRepository.create([
21             { name, password: passwordHash, email, driver_license }
22         ]);
23     }
24 }
```

No authentication importa compare do bcryptjs

```
const passwordMatch = await compare(password, user.password);

if (!passwordMatch) {
    throw new Error("Email or password incorrect!");
}
```

Validando email

```
src > modules > accounts > repositories > IUsersRepository.ts > ...
1  import { ICreateUserDTO } from "../dtos/ICreateUserDTO";
2  import { User } from "../entities/User";
3
4
5  interface IUsersRepository {
6      create(data: ICreateUserDTO): Promise<void>;
7      findByEmail(email: string): Promise<User>;
8  }
```

userRepository

```
2
3   async findByEmail(email: string): Promise<User> {
4       const user = await this.repository.findOne({ email });
5
6       return user;
7   }
8 }
```

Usecase

```
async execute({ name, password, email, driver license }: ICreateUserDTO): Promise<void> {
    const userAlreadyExists = await this.usersRepository.findByEmail(email);

    if (userAlreadyExists) {
        throw new Error("User already exists!");
    }
}
```

JWT JSON WEB TOKEN [yarn add jsonwebtoken yarn add @types/jsonwebtoken -D](#)

Dentro do usecase cria pasta authenticateUser e cria o UseCase e o Controller.

No useCase

No Controller

```
interface IResponse {
    user: {
        name: string;
        email: string;
    },
    token: string;
}
```

Usecase authentication

```

async execute({ email, password }: IRequest): Promise<IResponse> {
  // usuario existe
  const user = await this.usersRepository.findByEmail(email);

  // senha está correta
  if (!user) {
    throw new Error("Email or password incorrect!");
  }

  const passwordMatch = await compare(password, user.password);

  if (!passwordMatch) {
    throw new Error("Email or password incorrect!");
  }

  // 1 = payload info n critica, 2 = palavrasedreta 'jromarioss', 3 = obj
  const token = sign({}, "c82fdf0ebf03c6e8494030621c6d21b3", {
    subject: user.id,
    expiresIn: "1d"
  });

  const TokenReturn: IResponse = {
    token,
    user: {
      name: user.name,
      email: user.email
    }
  };

  return TokenReturn;

  // gerar o jsonwebtoken
}

```

Controller

```

modules > accounts > useCases > authenticateUser > AuthenticateUserController.ts > AuthenticateUserController > handle
1  import { Request, Response } from "express";
2  import { container } from "tsyringe";
3
4  import { AuthenticateUserUseCase } from "../authenticateUserUseCase";
5
6
7  class AuthenticateUserController {
8    async handle(request: Request, response: Response): Promise<Response> {
9      const { password, email } = request.body;
10
11      const authenticateUserUseCase = container.resolve(AuthenticateUserUseCase);
12
13      const token = await authenticateUserUseCase.execute({ email, password });
14
15      return response.json(token);
16    }
17  }
18
19  export { AuthenticateUserController }

```

E cria rotas authentication

```

routes > authentication > routes.ts
1  import { Router } from "express";
2  import { AuthenticateUserController } from "../modules/accounts/useCases/authenticateUser/AuthenticateUserController";
3
4  const authenticateRoutes = Router();
5
6  const authenticateUserController = new AuthenticateUserController();
7
8  authenticateRoutes.post("/sessions", authenticateUserController.handle);
9
10 export { authenticateRoutes };

```


E no index de rotas

```
3 router.use(authenticateRoutes);
4
```

Rotas autenticadas criar um middleware para verificar se é um token valido.

Funciona com o Beare Token

```
    async findById(id: string): Promise<User> {
      const user = await this.repository.findOne(id);
      return user;
    }
  }
}
```

```
5 interface IUsersRepository {
6   create(data: ICreateUserDTO): Promise<void>;
7   findByEmail(email: string): Promise<User>;
8   findById(id: string): Promise<User>;
9 }
10
11 export { IUsersRepository };
```

Cria pasta middleware

```
> middleware > ensureAuthenticated.ts > ensureAuthenticated > user
1 import { NextFunction, Request, Response } from "express";
2 import { verify } from "jsonwebtoken";
3
4 import { UsersRepository } from "../modules/accounts/repositories/implementations/UsersRepository";
5
6 interface IPayload {
7   sub: string;
8 }
9
10 export async function ensureAuthenticated(request: Request, response: Response, next: NextFunction) {
11   // Bearer token dawdad vem dentro do headers o token vem dentro do headers
12   const authHeader = request.headers.authorization;
13
14   if (!authHeader) { // se o header vem vazio
15     throw new Error("Token missing");
16   }
17
18   // Bearer token dawdadwfsefse o split divide pelo espaço, [bearer, token] pega posicao 2
19   const [, token] = authHeader.split(" ");
20
21   try {
22     // sub é o id do user
23     const { sub: user_id } = verify(token, "c82fdf0ebf03c6e8494030621c6d21b3") as IPayload;
24
25     // verificar se o usuário existe
26     const userRepository = new UsersRepository();
27
28     const user = await userRepository.findById(user_id);
29
30     if (!user) {
31       throw new Error("User does not exists!");
32     }
33
34     next();
35   } catch (error) {
36     throw new Error("Invalid token!");
37   }
38 }
```

Alterar tabelas

```

> database > migrations > 1669979467860-AlterUserDeleteUsername.ts > AlterUserDeleteUsername1669979467860
1 import { MigrationInterface, QueryRunner, TableColumn } from "typeorm";
2
3 export class AlterUserDeleteUsername1669979467860 implements MigrationInterface {
4
5     public async up(queryRunner: QueryRunner): Promise<void> {
6         await queryRunner.dropColumn("users", "username");
7     }
8
9     public async down(queryRunner: QueryRunner): Promise<void> {
10         await queryRunner.addColumn("users",
11             new TableColumn({
12                 name: "username",
13                 type: "varchar"
14             })
15         );
16     }
17 }
18

```

Cria pasta erros

```

> errors > AppError.ts > AppError > constructor
1 export class AppError {
2     public readonly message: string;
3
4     public readonly statusCode: number;
5
6     constructor(message: string, statusCode = 400) {
7         this.message = message;
8         this.statusCode = statusCode;
9     }
10 }

```

E coloca appero em tudo e o statuscode

```

if (!authHeader) { // se o ehader vem vazio
    throw new AppError("Token missing", 401);
}

```

E no server dps ds rotas

```

9
10 app.use((err: Error, request: Request, response: Response, next: NextFunction) => {
11     if (err instanceof AppError) {
12         return response.status(err.statusCode).json({
13             message: err.message
14         });
15     }
16
17     return response.status(500).json({
18         status: "error",
19         message: `Internal server error - ${err.message}`
20     });
21 });
22
23 app.listen(3333, () => {
24     console.log("Servis is running!");
25 });

```

E instalar [yarn add express-async-errors](#)

```
1 import ReflectMetadata ;
2 import express, { NextFunction, Request, Response } from "express";
3 import "express-async-errors";
4 import swaggerUi from "swagger-ui-express";
5
6 import { router } from "./routes";
```

Avatar

```
database > migrations > 1670062489492-AlterUserAddAvatar.ts > AlterUserAddAvatar1670062489492 > down
1 import {MigrationInterface, QueryRunner, TableColumn} from "typeorm";
2
3 export class AlterUserAddAvatar1670062489492 implements MigrationInterface {
4
5     public async up(queryRunner: QueryRunner): Promise<void> {
6         await queryRunner.addColumn("users",
7             new TableColumn({
8                 name: "avatar",
9                 type: "varchar",
10                nullable: true
11            })
12        )
13    }
14
15     public async down(queryRunner: QueryRunner): Promise<void> {
16         await queryRunner.dropColumn("users", "avatar");
17     }
18 }
19
20 }
```

Usecase

```

import { inject, injectable } from "tsyringe";
import { IUsersRepository } from "../../repositories/IUsersRepository";

interface IRequest {
  user_id: string;
  avatar_file: string;
}

@Inject()
class UpdateUserAvatarUseCase {
  constructor(
    @inject("UsersRepository")
    private usersRepository: IUsersRepository
  ) {}

  async execute({ user_id, avatar_file }: IRequest): Promise<void> {
    const user = await this.usersRepository.findById(user_id);

    user.avatar = avatar_file;

    await this.usersRepository.create(user);
  }
}

export { UpdateUserAvatarUseCase }

```

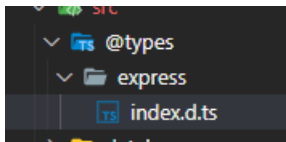
Controller

```

modules > accounts > useCases > updateUserAvatar > UpdateUserAvatarController.ts > ...
1 import { Request, Response } from "express";
2 import { container } from "tsyringe";
3
4 import { UpdateUserAvatarUseCase } from "../UpdateUserAvatarUseCase";
5
6
7 class UpdateUserAvatarController {
8   async handle(request: Request, response: Response): Promise<Response> {
9     const { id } = request.user;
10
11     // receber arquivo
12     const avatar_file = null;
13
14     const updateUserAvatarUseCase = container.resolve(UpdateUserAvatarUseCase);
15
16     await updateUserAvatarUseCase.execute({ user_id: id, avatar_file});
17
18     return response.status(200).send();
19   }
20 }
21
22 export { UpdateUserAvatarController }

```

Criar a pasta, sobreescrever types



```
@types > express > index.d.ts > {} Express
1 declare namespace Express {
2   export interface Request {
3     user: {
4       id: string;
5     }
6   }
7 }
```

Criar pasta config e upload.ts

```
config > upload.ts > ...
import multer from "multer";
import { resolve } from "path";
import crypto from "crypto";

export default {
  upload(folder: string) {
    return {
      storage: multer.diskStorage({
        destination: resolve(__dirname, "..", "..", folder),
        filename: (request, file, callback) => {
          const fileHash = crypto.randomBytes(16).toString("hex");
          const fileName = `${fileHash}-${file.originalname}`;

          return callback(null, fileName);
        }
      })
    }
  }
}
```

Nas rotas

```
import { UpdateUserAvatarController } from "../modules/accounts/userController";
import uploadConfig from "../config/upload";

const usersRoutes = Router();

const uploadAvatar = multer(uploadConfig.upload("./tmp/avatar"));

const createUserController = new CreateUserController();

16
17 usersRoutes.patch("/avatar", uploadAvatar.single("avatar"), updateUserAvatarController.handle);
18
19 export { usersRoutes }
```

E no controller

```
class UpdateUserAvatarController {
  async handle(request: Request, response: Response): Promise<Response> {
    const { id } = request.user;
    // receber arquivo
    const avatar_file = request.file.filename;
  }
}
```

Deletando arquivo existente

```
src > config > file.ts > deleteFile
1  import fs from "fs";
2
3  export const deleteFile = async(filename: string) => {
4    // stat verifica se um arquivo existe
5    try {
6      await fs.promises.stat(filename);
7    } catch(err) {
8      return;
9    }
10
11    // responsavel por remover o arquivo
12    await fs.promises.unlink(filename);
13  }
14 }
```

E no usecase do updateavatar

```
async execute({ user_id, avatar_file }: IRequest): Promise<void> {
  const user = await this.usersRepository.findById(user_id);

  if (user.avatar) {
    await deleteFile(`./tmp/avatar/${user.avatar}`);
  }
}
```

Testes

Instalar jest [yarn add jest @types/jest -D yarn ts-jest -D](#)

Yarn jest --init

```
// A preset that is
preset: "ts-jest",

// The glob patterns Jest uses to
testMatch: ["**/*.spec.ts"],

// Stop running
bail: true,
```

```
// agrupa os teste
describe("Criar categoria", () => {
  it("Espero que 2 + 2 seja 4", () => {
    const soma = 2 + 2;
    const resultado = 4;

    expect(soma).toBe(resultado);
  });

  it("Espero que 2 + 2 não seja 5", () => {
    const soma = 2 + 2;
    const resultado = 5;

    expect(soma).not.toBe(resultado);
  })
});
```

No repositories cria pasta in-memory para fazer criação na memória para não fazer teste no banco de dados pq não é a responsabilidade dos testes unitários.

```
modules > cars > repositories > in-memory > CategoriesRepositoryInMemory.ts > CategoriesRepositoryInMemory > findByName
import { Category } from "../../entities/Category";
import { ICategoriesRepository, ICreateCategoryDTO } from "../ICategoriesRepository";

class CategoriesRepositoryInMemory implements ICategoriesRepository {
  categories: Category[] = [];

  async findByName(name: string): Promise<Category> {
    const category = this.categories.find(category => category.name === name);
    return category;
  }

  async list(): Promise<Category[]> {
    const list = this.categories;
    return list;
  }

  async create({ name, description }: ICreateCategoryDTO): Promise<void> {
    const category = new Category();

    Object.assign(category, {
      name, description
    });

    this.categories.push(category);
  }
}

export { CategoriesRepositoryInMemory }
```



```

1 import { AppError } from "../../errors/AppError";
2 import { CategoriesRepositoryInMemory } from "../../repositories/in-memory/CategoriesRepositoryInMemory";
3 import { CreateCategoryUseCase } from "../CreateCategoryUseCase";
4
5 let createCategoryUseCase: CreateCategoryUseCase;
6 let categoriesRepositoryInMemory: CategoriesRepositoryInMemory;
7
8 // agrupa os teste
9 describe("Create Category", () => {
10
11     // faça antes do teste
12     beforeEach(() => {
13         categoriesRepositoryInMemory = new CategoriesRepositoryInMemory();
14         createCategoryUseCase = new CreateCategoryUseCase(categoriesRepositoryInMemory);
15     })
16
17     it("should be able to create a new category", async () => {
18         const category = {
19             name: "Category Test",
20             description: "Category description Test",
21         }
22
23         await createCategoryUseCase.execute({
24             name: category.name,
25             description: category.description
26         });
27
28         const categoryCreated = await categoriesRepositoryInMemory.findByName(category.name);
29
30         // se tiver a propriedade id foi criada
31         expect(categoryCreated).toHaveProperty("id");
32     });
33
34     it("should not be able to create a new category with the same name", async () => {
35         expect(async () => {
36             const category = {
37                 name: "Category Test",
38                 description: "Category description Test",
39             }
40
41             await createCategoryUseCase.execute({
42                 name: category.name,
43                 description: category.description
44             });
45
46             await createCategoryUseCase.execute({
47                 name: category.name,
48                 description: category.description
49             });
50         }).rejects.toBeInstanceOf(AppError)
51     });
52

```

Teste user

```

import { ICreateUserDTO } from "../../dtos/ICreateUserDTO"
import { User } from "../../entities/User"
import { IUsersRepository } from "../IUsersRepository"

class UsersRepositoryInMemory implements IUsersRepository {
  users: User[] = [];

  async create({ email, name, password, driver_license }: ICreateUserDTO): Promise<void> {
    const user = new User();

    Object.assign(user, {
      email, name, password, driver_license
    });

    this.users.push(user)
  }

  async findByEmail(email: string): Promise<User> {
    return this.users.find((user) => user.email === email);
  }

  async findById(id: string): Promise<User> {
    return this.users.find((user) => user.id === id);
  }
}

export { UsersRepositoryInMemory }

```

Teste de criar o usuário

```

let authenticateUserUseCase: AuthenticateUserUseCase;
let usersRepositoryInMemory: UsersRepositoryInMemory;
let createUserUseCase: CreateUserUseCase;

describe("Authenticate User", () => {
  beforeEach(() => {
    usersRepositoryInMemory = new UsersRepositoryInMemory();
    authenticateUserUseCase = new AuthenticateUserUseCase(usersRepositoryInMemory);
    createUserUseCase = new CreateUserUseCase(usersRepositoryInMemory);
  });

  it("should be able to authenticate a user", async () => {
    // criando usuario
    const user: ICreateUserDTO = {
      driver_license: "000123",
      email: "user@teste.com",
      password: "12345",
      name: "User Teste",
    }

    await createUserUseCase.execute(user);

    const result = await authenticateUserUseCase.execute({
      email: user.email,
      password: user.password,
    });

    // mostra oque irá retornar, que é o token
    // console.log(result);

    expect(result).toHaveProperty("token");
  });
});

```

Verificação se n existe o email e a senha errado.

```

it("should not be able to authenticocate an none existent user", () => {
  expect(async () => {
    await authenticateUserUseCase.execute({
      email: "false@email.com",
      password: "1234",
    });
  }).rejects.toBeInstanceOf(AppError);
});

it("should not be able to authenticate with incorrect password", () => {
  expect(async () => {
    const user: ICreateUserDTO = {
      driver_license: "99999",
      email: "user@user.com",
      password: "22123",
      name: "User teste error",
    }

    await createUserUseCase.execute(user);

    await authenticateUserUseCase.execute({
      email: user.email,
      password: "incorrectpassword",
    });
  }).rejects.toBeInstanceOf(AppError);
});

```

Automatizando os imports no tsconfig procure por:

```

/* Modules */
"module": "commonjs",
// "rootDir": "./",
// "moduleResolution": "node",
"baseUrl": "./src",
"paths": {
  "@modules/*": ["modules/*"],
  "@config/*": ["config/*"],
  "@shared/*": ["shared/*"],
  "@errors/*": ["errors/*"],
},
// "rootDirs": [],

```

Baixar para traduzir as importações [yarn add tsconfig-paths -D](#)

E no package.json colocar -r tsconfig-paths/register

```

  "scripts": {
    "dev": "ts-node-dev -r tsconfig-paths/register --inspect --",
    "typeorm": "ts-node-dev -r tsconfig-paths/register ./node_m",
    "test": "jest"
  }

```

Configurando os testes vai no jestconfig:

```
jest.config.ts •
jest.config.ts > [default]

1 import { pathsToModuleNameMapper } from "ts-jest";
2 import { compilerOptions } from "./tsconfig.json";
3
module
moduleNameMapper: pathsToModuleNameMapper(compilerOptions.paths, {
  prefix: "<rootDir>/src/",
}),
```

****RF**** => Requisitos funcionais

****RNF**** => Requisitos não funcionais

****RN**** => Regra de negócio

Chave estrangeira: fecha as colunas

```
name: "cars",
columns: [ ...
],
foreignKeys: [
  {
    name: "FKCategoryCar",
    referencedTableName: "categories",
    referencedColumnNames: ["id"],
    columnNames: ["category_id"],
    onDelete: "SET NULL",
    onUpdate: "SET NULL"
  }
]
```

No entitites em car.

```

@Column()
brand: string;

@ManyToOne(() => Category)
@JoinColumn({ name: "category_id"})
category: Category;

@Column()
category_id: string;

```

Criando usuário adm

```

shared > infra > typeorm > seed > admin.ts > ...
import { hash } from "bcryptjs";
import { v4 as uuidV4 } from "uuid";

import createConnection from "shared/infra/typeorm/index";

async function create() {
  const connection = await createConnection("localhost");

  const id = uuidV4();
  const password = await hash("admin", 8);

  await connection.query(
    `INSERT INTO USERS(id, name, email, password, "isAdmin", created_at, driver_license)
    values('${id}', 'admin', 'admin@admin.com.br', '${password}', 'true', 'now()', 'XXXXXX')`
  );

  await connection.close();
}

create().then(() => console.log("User admin created!"));

```

```
"seed:admin": "ts-node-dev -r tsconfig-paths/register src/shared/infra/typeorm/seed/admin.ts"
```

Middle de admin

```

src > shared > infra > http > middlewares > ensureAdmin.ts > ...
1  import { NextFunction, Request, Response } from "express";
2
3  import { AppError } from "@shared/errors/AppError";
4  import { UsersRepository } from "@modules/accounts/infra/typeorm/repositories/UsersRepository";
5
6  export async function ensureAdmin(request: Request, response: Response, next: NextFunction) {
7    const { id } = request.user;
8
9    const usersRepository = new UsersRepository();
10   const user = await usersRepository.findById(id);
11
12   if (!user.isAdmin) {
13     throw new AppError("User isn't admin!");
14   }
15
16   return next();
17 }

```

Criando tabela com chave estrangeira

```

export class CreateSpecificationsCars1670495130974 implements MigrationInterface {

  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.createTable(
      new Table({
        name: "specifications_cars",
        columns: [
          {
            name: "card_id",
            type: "uuid"
          },
          {
            name: "specification_id",
            type: "uuid"
          },
          {
            name: "created_at",
            type: "timestamp",
            default: "now()"
          }
        ]
      })
    );

    await queryRunner.createForeignKey(
      "specifications_cars",
      new TableForeignKey({
        name: "FKSpecificationCar",
        referencedTableName: "specifications",
        referencedColumnNames: ["id"],
        columnNames: ["specification_id"],
        onDelete: "SET NULL",
        onUpdate: "SET NULL"
      })
    );

    public async down(queryRunner: QueryRunner): Promise<void> {
      await queryRunner.dropForeignKey("specifications_cars", "FKSpecificationCar");

      await queryRunner.dropForeignKey("specifications_cars", "FKCarSpecification");

      await queryRunner.dropTable("specifications_cars")
    }
  }
}

```

Quando tem tabela de relacionamento é many to many.

Caso der esse error, Cannot read properties of undefined (reading 'findByIds'), oque vem antes do metodo vem nulo.

```

beforeEach(() => {
  carsRepositoryInMemory = new CarsRepositoryInMemory()
  specificationRepositoryInMemory = new SpecificationsRepositoryInMemory();
})

```

Trabalhar com data [yarn add dayjs](#)

```

> modules > rentals > usecase > createRental > CreateRentalsUse
1  import dayjs from "dayjs";
2  import utc from "dayjs/plugin/utc"; 8  dayjs.extend(utc);
3
9

```

```
import dayjs from "dayjs";
import utc from "dayjs/plugin/utc";

import { IDateProvider } from "../IDateProvider";

dayjs.extend(utc);

class DayjsDateProvider implements IDateProvider {

  compareInHours(start_date: Date, end_date: Date): number {
    const end_date_utc = this.convertToUtc(end_date);
    const start_date_utc = this.convertToUtc(start_date);

    return dayjs(end_date_utc).diff(start_date_utc, "hours"); // converted a comparação
  }

  convertToUtc(date: Date): string {
    return dayjs(date).utc().local().format(); // gera data baseada utc
  }

  dateNow() {
    return dayjs().toDate();
  }
}

export { DayjsDateProvider }
```

```
interface IDateProvider {
  compareInHours(start_date: Date, end_date: Date): number;
  convertToUtc(date: Date): string;
  dateNow(): Date;
}

export { IDateProvider }
```

```
// - o aluguel deve ter duração mínima de 24 horas.
const dateNow = this.dateProvider.dateNow();

const compare = this.dateProvider.compareInHours(dateNow, expected_return_date);

if (compare < minimumHour) {
  throw new AppError("Invalid return time!");
}
```

SUPER TEST yarn add supertest e o @types

```
"scripts": {
  "dev": "ts-node-dev -r tsconfig-paths/register src/index.ts",
  "typeorm": "ts-node-dev -r tsconfig-paths/register src/index.ts",
  "test": "NODE_ENV=test jest",
}
```



```
src > shared > infra > typeorm > index.ts > default
15 export default async(host = "database"): Promise<Connection> => {
16   const defaultOptions = await getConnectionOptions();
17
18   return createConnection(
19     Object.assign(defaultOptions, {
20       host: process.env.NODE_ENV === "test" ? "localhost" : host,
21       database: process.env.NODE_ENV === "test" ? "rentx_test" : defaultOptions.database,
22     })
23   );
24 }
```

Criar um database de teste

```
3
4 create database rentx_test
5
6
```

```
modules > cars > useCases > createCategory > CreateCategoryController.spec.ts > describe("Create Category Controller") callback
1 import request from "supertest";
2 import { Connection } from "typeorm";
3 import { v4 as uuidV4 } from "uuid";
4 import { hash } from "bcryptjs";
5
6 import { app } from "@shared/infra/http/app";
7
8 import createConnection from "@shared/infra/typeorm";
9
10 let connection: Connection;
11
12 describe("Create Category Controller", () => {
13   beforeAll(async () => {
14     connection = await createConnection();
15     await connection.runMigrations();
16
17     const id = uuidV4();
18     const password = await hash("admin", 8);
19
20     await connection.query(
21       `INSERT INTO USERS(id, name, email, password, "isAdmin", created_at, driver_license)
22       values('${id}', 'admin', 'admin@admin.com.br', '${password}', 'true', 'now()', 'XXXXXX')`
23     );
24   });
25 }
```

```

afterAll(async () => {
  await connection.dropDatabase();
  await connection.close();
})

it("should be able to create a new category", async () => {
  const responseToken = await request(app).post("/sessions")
    .send({
      email: "admin@admin.com.br",
      password: "admin"
    });

  const { token } = responseToken.body;

  const response = await request(app)
    .post("/categories")
    .send({
      name: "Category supertest",
      description: "supertest category"
    }).set({
      Authorization: `Bearer ${token}`,
    });

  expect(response.status).toBe(201);
});

```

```

it("should be able to list all categories", async () => {
  const responseToken = await request(app).post("/sessions")
    .send({
      email: "admin@admin.com.br",
      password: "admin"
    });

  const { token } = responseToken.body;

  await request(app)
    .post("/categories")
    .send({
      name: "Category supertest",
      description: "supertest category"
    }).set({
      Authorization: `Bearer ${token}`,
    });

  const response = await request(app).get("/categories");

  console.log(response.body);

  expect(response.status).toBe(200);
  expect(response.body.length).toBe(1);
  expect(response.body[0]).toHaveProperty("id");
  expect(response.body[0].name).toEqual("Category supertest");
});

```

